A Novel Dynamical Approach to (m,k)-firm Scheduling

Milton Armando Cunguara,Tomás António Mendes Oliveira e Silva
DETI/IEETA/University of Aveiro
Aveiro, Portugal
{milton.cunguara,tos}@ua.pt

Paulo Bacelar Reis Pedreiras
DETI/IT/ University of Aveiro
Aveiro, Portugal
pbrp@ua.pt

***Abstract*—Many real-time systems tolerate, up to some extent, that instances of their tasks are skipped or miss their deadlines. In such cases it is possible to increase the system schedulability by dropping some jobs. However, the number and temporal distribution of such drops must be constrained, in order to bound its impact on the system performance. The (m,k)–firm model, which specifies that at least m out of any k consecutive jobs are scheduled, captures such constraints and has been a topic of interest of the scientific community over the last few years. Many advances in the implementation of this type of schedulers were made. However, the (m,k)–firm schedulers hitherto proposed in literature either do not provide deterministic (m,k)–firm guarantees or are not efficiently scheduled. In this paper it is presented a (m,k)–firm scheduler that provides deterministic (m,k)–firm guarantees and is capable of scheduling task sets that are deemed unschedulable by current approaches. The proposed scheduler works by dynamically changing the underlying (m,k)–firm frames. Examples of task sets that are schedulable by the scheduler proposed in this paper but that not schedulable by any other scheduler described in the literature are presented.**

I. Introduction

Many real-time systems allow occasional deadline violations or even the non-execution of some of its task's jobs. Classical examples of such systems are encoded video processing and control applications. A simple form of specifying the degree of tolerance of an application to such faults consists in the indication of an average fault ratio. Tough simple, such specification is not rich enough, since it does not capture the temporal distribution of the faults. For instance, most control applications are robust enough to cope with the omission of one sensor sample every ten sampling periods, without incurring a serious performance degradation. However, such systems may easily become unstable if, during a window of 100 sampling periods, 10 consecutive sensor omissions are experienced. To address this lack of expressiveness, it was proposed in the literature the (m,k)–firm model, which indicates that at least m out of any k consecutive jobs shall be correctly executed. In addition to the specification of an acceptable fault ratio, this model constraints the maximum time interval between any two consecutive correct jobs. This increased degree of expressiveness suits the needs of many application domains and, for this reason, the (m,k)-firm model has been a topic of interest of the scientific community over the last few years.

(m,k)–firm schedulers can be classified either as **probabilistic** or **deterministic**. **Probabilistic** (m,k)–firm schedulers try to reduce the amount of time in which the (m,k)–firm guarantees are not met, but its primary goal is to ensure that **on average** m out of k job executions are met. Such approach may have deleterious effects on the scheduled tasks since does not bound the number of consecutive jobs that may be skipped. On the other hand, **deterministic** (m,k)–firm schedulers guarantee that for any group of k consecutive jobs at least m are executed. However, the deterministic schedulers that have been presented so far in the literature suffer from a common illness, which consists in the need to actually reserve computational resources according with a (k,k)–firm requirement, when worst-case conditions are considered [1]. Therefore, in this scenario, there is no effective reduction on processing requirements.

This paper presents a new (m,k)–firm scheduler that does not require a (k,k)–firm reservation, and thus that is able to schedule task sets that are unschedulable by classical (m,k)–firm schedulers. The scheduler is dynamic in the sense that the selection of which jobs are executed does not depend only on the priority of the tasks to which they belong, but also of the past executions. In a certain sense it can be said that priorities are associated with jobs as opposed to tasks. It is shown that all task sets that are schedulable by classical schedulers are also schedulable by the scheduler presented herein, while the opposite is not true. Furthermore, the scheduler proposed in this paper allows attaining higher utilizations for task sets that are schedulable by classical (m,k)–firm schedulers.

This paper is organized as follows: Section II presents a review of the state of the art of the field. Section III revisits the static (m,k)–firm frames, which are presented under a new light that leads to the main contribution of the paper, which is presented in Section IV. Section V contains a number of examples that allow a better understanding of the working principles of the proposed scheduler. Section VI makes a comparison of the new scheduler with classical ones. Section VII presents the concluding remarks.

This work was partially supported by the Portuguese Government through FCT - Fundacao para a Ciencia e a Tecnologia in the scope of project Serv-CPS -PTDC/EEA-AUT/122362/2010 and Ph.D. grant - SFRH/BD/62178/2009/J53651387T4E

978-1-4799-3235-1/14/$31.00 ©2014 IEEE

II. Related Work

In [2] it is proposed a scheduler that divides the task's jobs into mandatory and optional ones. Each frame of k jobs should have m mandatory jobs. Jobs are set as mandatory or optional based on an upper mechanical word. The generated frames have an evenly distributed mandatory activation pattern. In [3], it was proposed one of the first alternative mandatory job assignment scheme. A similar contribution was given in [4], in which it was proposed that the mandatory job assignments should minimize a certain metric, dubbed Granularity of Quality of Service-Reward (GQoS-reward). These approaches were in total contrast with the use of distance to failure, which was the *de facto* approach to realize (m,k)–firm schedulers.

In [5], it was proved that the scheduler in [2] introduces a periodic critical moment, one of which is at $t = 0$. Thus, the even distribution of mandatory jobs is not necessarily the best. Three different algorithms for finding a suitable schedule for the mandatory tasks, that differ primarily in their computational efficiency, were propose. The first resembles a test of all possible schedules, the second is a genetic algorithm and the third one introduces a metric that drives the search parameters of the genetic algorithm. Notwithstanding the exhaustive nature of the proposed algorithm, in general it fails to reach the optimal schedule because, as will be shown latter on in this paper, the search space of the algorithm is smaller than the solution space. Furthermore, the *solutions* that the paper proposes are still haunted by the problem that it intended to solve, namely, the proposed scheduler also introduces periodical critical instants.

A different type of static pattern was presented in [6], the so called Deeply Red (or R-) pattern, which is defined by setting all jobs that verify $j_i \bmod k_i < m_i$ as mandatory, where j_i is an integer that is incremented by one whenever a job from the τ_i is released. If $j_i \bmod k_i = 0$ at the release of the first job of the task, then the generated sequence will consist of m_i mandatory jobs in the beginning of the cycle. j_i introduces a phase, but it is impossible to escape from a critical instant in the general case, where there is no special relation between the task's periods.

It was proved in [7] that if a given (m,k)–firm task set is schedulable using a Deeply Red pattern then it is schedulable using any other static k jobs frame with m job activations. This is not the same as optimality, which implies that if exists a scheduler that can schedule a given system then the proposed scheduler can also schedule the system in question. In fact, [7] proves that there is no static (m,k)–firm scheduler that performs worse than the Deeply Red pattern.

In [1] it is noted this limitation regarding the provisioning of deterministic (m,k)–firm guarantees using static frames of length k with m job activations and is argued that the deterministic approach "*does not contribute to reducing the resource requirements in general*". Due to 1) static frames introduce critical instants, 2) tasks with implicit deadlines can be scheduled on a critical instant if the task set is also EDF-schedulable, therefore, 3) all task sets that provide deterministic guarantees are EDF-schedulable. However, this reasoning is based on the assumption that [2], [5] and other contemporary (m,k)–firm schedulers are optimal. But, as will be shown, contemporary schedulers are suboptimal since they introduce critical instants.

Regarding the implementation end of the field, one of the first papers to tackle the problem of (m,k)–firm scheduling [8] presented a scheme in which messages were supposed to achieve probabilistic (m,k)–firm guarantees by attributing to each task a *distance to failure* with higher priorities being attributed to tasks with lower distances, i.e., close to failure. The distance to failure is the number of future consecutive jobs that had to not be executed in order to put the task in a dynamical failure. In case of a draw, i.e., several tasks with the same difference to failure, the choice is EDF based. This approach was dubbed Distance Based Priority (DBP). A similar approach was put forth in [9], in which an (m,k)–firm model for wireless networks was presented. It had some aspects relevant to wireless networks, e.g., the distance to the sink (signal strength). A similar probabilistic approach was first proposed in [10].

Other variants of DBP include Integrated DBP (IDBP) [11], which also used the distance to recovery upon failure, that significantly reduces the *domino effect* in which, as in EDF, the failure in one job/task is propagated into other jobs/tasks. Matrix-DBP [12] also uses the distance to failure to schedule message streams, but differs at a fundamental level from other variants since it cannot be preemptive. It employed a rather large set of other variables such as periods, service (execution) time and relationship between streams. An expanded version of the matrix-DBP into the (E_matrix) DBP, which allows for non-periodical messages, was presented in [12]. Similarly, [13] improves upon DBP by introducing Total BDP (TDBP) which introduces a *total distance* to failure that include other relevant parameters, such as distance to enter invalidation (dynamic failure) and to exit it, which in much resembles the goals of IDBP [11].

In [14], it is presented a multiprocessor implementation of a protocol of the DBP family, as well as an analysis of its probability of being in a dynamic failure. In [15] it is proposed a protocol called Local DBP (L-DBP) which has the goal of providing (m,k)–firm guarantees to multimedia streams over wireless sensor networks. It works by augmenting the distance of DBP with local node information regarding link congestion and failure rates.

In [16], it is discussed the possibility of performing an online schedulability test that admits optional jobs if and only if it can be assured that they will not cause a mandatory job to miss a deadline. This is further enforced by putting the jobs into two different execution queues: a high priority queue for mandatory jobs and a low priority one for optional jobs, with jobs from the low priority queue executing if and only if the high priority queue is empty. The proposed schedulability test is not tight. The approach presented in this paper also uses a schedulability test. However, the proposed approach is less restrictive in the sense that optional jobs are allowed to

interfere with mandatory ones for as long as they do not cause a deadline miss.

In [17] it is presented the approach most similar to the one presented herein. The authors use the time to failure combined with deadline awareness to mitigate some of the issues related to the original DBP. Two algorithms are presented. The guaranteed on-line scheduling algorithm (GDPA) is the most powerful and complex one, while the other one is a lightweight version of it. In GDPA it is decided to put a given job in the execution queue based on the local (temporal) state of the queue. Thus, it can put jobs that have long distances to failure in the queue, as long as there are no jobs of tasks with lower distances at that moment. However, it is possible that in the following scheduling instants jobs from tasks close to a dynamic failure are released and eventually may fail their deadline. An example of this effect is explored in-depth in Section V.

Applications of (m,k)–firm scheduling in industrial control are presented, for example, in [18] in which the fact that many control applications are resilient to a small number of failures is used to reduce the resource usage. Controller modifications are also discussed. In [19] the graceful QoS degradation provided by (m,k)–firm scheduling is explored. More concretely, DBP is analyzed and sufficient, though not tight, conditions for deterministic schedulability and their applications for control purposes were proposed. [20] tries to devise an optimal (m,k)–firm frame for control purposes. However, no consideration regarding the schedulability of the resulting (m,k)–firm sequence was made, which could invalidate the underlying assumptions. In fact, this is a general issue regarding the utilization of contemporary (m,k)–firm schedulers for industrial applications.

In [21] it is presented a study of the application of (m,k)–firm schedulers in multimedia settings, with a special focus on DBP. Multimedia is one of the areas in which (m,k)–firm scheduling has the potential to provide considerable improvements, as can be attested in [21] and references therein. For example, for the JPEG video case, there are two types of frames. The most important ones are designated key frames and are used to decode the other less important frames. Thus, key frames must always be decoded, which conflicts with many dynamic schedulers, because dynamic schedulers do not provide guarantees regarding which job will be executed. For example, DBP only considers the distance to failure, neglecting the importance of the underlying frames.

III. Static (Circular) (m,k)-firm Schedulers

This section generalizes some theorems in the literature. The first theorem generalizes the theorem proved in [2] that states that upper mechanical (m,k)–firm frames generate (m,k)–firm sequences that always meet the respective guarantees. The theorem is generalized for any frame with the respective size and number of mandatory jobs. The second theorem helps to understand the periodic nature of (m,k)–firm sequences. In [5] is proven that the use of the Chinese reminder algorithm can be used to schedule (m,k)–firm task sets. The theorem is generalized to allow any algorithm that handles periodically repeating activation patterns. This is of uttermost importance because the Chinese reminder algorithm is known to be NP-hard, i.e., it cannot be solved in polynomial time.

These theorems will in turn be fundamental at establishing the improvements of the dynamical scheduler proposed in this paper. But before that, a number of definitions are in order.

Definition. *An* (m,k)–firm *frame is any group of* k *bits in which* m *and only* m *bits are ones.*

Definition. *A job is optional if the rightmost bit of its* (m,k)–firm *frame is zero, otherwise the job is mandatory.*

Definition. *An* (m,k)–firm *sequence is any sequence of job executions in which for any group of* k *consecutive job activations, at least* m *are executed.*

Furthermore, in this analysis it is made a number of assumptions:

- An Optional job is executed if and only if its execution does not cause the miss of a deadline of a mandatory job.
- An Optional job is executed if and only if it will be able to complete its execution within its respective deadline.
- Whenever a job is released, regardless of being executed or not, unless some pattern breaking event occurs, the whole (m,k)–firm frame associated with it is rotated one bit to the left.
- Tasks release jobs periodically, regardless of the job's priority.

The next theorem helps to understand the name (m,k)–firm frame

Theorem 1. *Any* (m,k)–firm *frame generates an* (m,k)–firm *sequence.*

Proof: The job execution sequence is encoded into the (m,k)–firm frame. So, if the sequence is rotated one bit to the left, then the bit sequence will be exactly the job execution sequence, where the newer job corresponds to the rightmost and the older one to the second from the right. Hence, in the worst case, when only mandatory jobs get executed, there will be exactly m jobs executed out of k, corresponding to the number of ones in the frame. ■

This theorem removes the distinctiveness of any special sequence, such as the upper mechanical word used in [2] and an intuition of it was the basis of the work in [5].

The next theorem helps to understand the static nature of (m,k)–firm frame.

Theorem 2. k *is a period of any task whose job sequence is generated by an* (m,k)–firm *frame.*

Proof: Given the way in which jobs are set as mandatory or optional, to prove the theorem, it suffices to prove the periodicity of the (m,k)–firm sequence. Now, since each job activation triggers one rotation of the (m,k)–firm frame, then k activations will trigger k rotations of the (m,k)–firm frame. Since the (m,k)–firm frame is k bits long, k rotations put it back in its original state. ■

Note, however, that k is not necessarily the smallest period, since the (m,k)–firm frame may have an *internal period*. Due to the periodicity, the phase is a number between 0 and $k-1$. The periodicity of the (m,k)–firm sequence has a strong bearing into the size of the search space, since if one (m,k)–firm frame can be rotated into another one, then they are the same (m,k)–firm frame with different phases. Nevertheless, since tasks sets normally have more than one task, it may happen that a given (m,k)–firm frame phase is feasible but some other phase of the same (m,k)–firm frame is not.

Now, consider the following definition, which helps to understand the ensemble of frames generated by the task set.

Definition. *An hiperframe is the shortest set of all job activations (mandatory/optional) and their respective order from all* (m,k)–firm *frames, between two distinct instants, such that in both instants all* (m,k)–firm *frames are in the same phase.*

As in the case of (m,k)–firm frames, one can also define a phase of the hiperframe which helps to identify its state at any given point of the period. Evidently, the search for a feasible (m,k)–firm schedule can be reduced to the search of a feasible hiperframe.

IV. Dynamic (m,k)-firm Schedules

Classical static schedulers repeat the schedules of each of their (m,k)–firm tasks. However, the best (m,k)–firm schedule does not necessarily belong to the set of schedules that can be generated by repeating (m,k)–firm frames, as is assumed, for example, in [5]. It is possible that the best schedule has the task set repeating itself with the periodicity of the hiperframe, as shown in the examples presented in Section V. For this reason, a different approach is proposed — Dynamic (m,k)–firm Schedulers.

Dynamic (m,k)–firm schedulers change the characteristics of the (m,k)–firm frames in such a way that optional jobs are executed, whenever possible, but without putting the task set in a situation in which it is not possible to meet the deadlines of all mandatory jobs.

These constant frame changes are what renders this schedule dynamic. Such changes to the (m,k)–firm frame must respect two conditions, namely:

- not violate any (m,k)–firm guarantees,
- improve the future schedulability of the remaining tasks.

To achieve the first point, it suffices to make transformations that 1) maintain the number of mandatory jobs (m) within the (k) bits of the frame, even if the new sequence is not a possible shift of the original one, and 2) it is done during an optional job that got executed. Theorem 3 proves that, under these two conditions, the (m,k)–firm guarantees are not violated.

One transformation that respects the above conditions is maintaining the rightmost bit in its position and rotate one bit to the left the resulting leftmost (m,k)–firm frame, as follows: $[b_{n-1}\, b_{n-2} \cdots b_1\, b_0] \rightarrow [b_{n-2}\, b_{n-3} \cdots b_1\, b_{n-1}\, b_0]$.

Theorem 3. *Let there be an* (m,k)–firm *task, belonging to a schedulable set of* (m,k)–firm *tasks. If after the execution of an optional job from a given task, the task in question maintains the most significant bit of its* (m,k)–firm *frame optional and rotates the remaining* $k-1$ *leftmost bits one bit to the left, then the resulting* (m,k)–firm *sequence will maintain its* (m,k)–firm *schedulability state.*

Proof: From the execution point of view, an optional task that gets executed is analogous to a mandatory job in the same point of the frame. In so being, consider Figure 1, in which a given frame is sampled. By assumption bit $b_0 = 0$, i.e., it is optional. However, in this execution sequence it was changed to $b_0 = 1$ since it was actually executed. Since, by assumption, the initial sequence respected the (m,k)–firm conditions, then after changing the last bit the initial k consecutive jobs contain $m+1$ executions. If the optional job does not get executed at the second pass, i.e., the second instance of b_0 then the following k consecutive jobs (starting at the first occurrence of b_{n-2}) get m executions from bits b_{n-2} to (the second) b_0 (if it also does not get executed). From there on, the (m,k)–firm guarantee would be met because it would be a static frame as presented before. ■

It must be stressed the importance of the last theorem, because it allows the execution of optional jobs without jeopardizing the (m,k)–firm guarantees. The execution of optional jobs was not considered in classical static (m,k)–firm schedulers and some version actually demanded that they did not take place at all, such as the upper mechanical and the Deeply Red sequences, whereas in probabilistic approaches there is more of a grey line in this issue due to the inherent differences in semantics.

A. Dynamic Scheduling of Optional Executions

Optional jobs can only be executed if its execution does not cause deadline violations of mandatory ones. Algorithm 1 allows to determine if a given optional job can be safely executed. Its operation is based on an *execution queue* that contains, in addition to the *ready* jobs, *close* future mandatory jobs. *Closeness* is defined as the time horizon in which the loading effect of the eventual introduction of an optional job disappears. The loading effect of an optional job is said to disappear at a given instant t_k if the finish times of all mandatory jobs that complete its execution at time $t \geq t_k$ are the same as if the optional job was not executed. To evaluate if optional jobs can be safely executed, the algorithm simulates its insertion in the execution queue and then evaluates if the deadlines of all the jobs in the execution queue (ready and close mandatory jobs) are violated.

$$\overbrace{[b_{n-1}\, b_{n-2} \cdots b_2\, b_1 \underbrace{b_0}_{b_0=0}}^{\mathrm{m+1\ executions}}{}^{rep\rightarrow}\, b_0\ b_{n-2}\, b_{n-3} \cdots b_1\, b_{n-1}\, b_0]$$

Fig. 1: State of (m,k)–firm frames during the execution of an optional job

By definition, optional jobs have lower priority than mandatory ones. However, when an optional job is inserted in the execution queue, its priority is virtually elevated, becoming the same as the mandatory jobs of the same task, and thus may preempt mandatory jobs of other tasks. Hence, the execution queue can be divided in two sets, one that contains only jobs with priority not lower than the priority of the job being tested (T^{HE}), and another one that contains jobs that have a priority lower than the one of the optional job being tested (T^{L}). The deadline of jobs in (T^{HE}) does not need to be evaluated as they are not affected by the insertion of the job being tested. The same is not true for jobs in T^{L}, as thus the deadlines of all jobs in this set, in conjunction with the optional job itself, must be evaluated.

Algorithm 1 has three chained repeat cycles. The outermost moves through the various jobs testing them for schedulability, the middle one moves through each job going through each T^{HE} and T^{L} windows. It sums up the T^{L} and stores in the c' variables to determine the total execution time. This variable is in turn used to determine the schedulability, i.e., whether or not it completes its WCET before its respective deadline. The innermost cycle is used to determine the duration of the T^{HE}, which is stored in the variable $c\_$.

The outermost cycle starts by setting the current time equal to the release time of the current process (τ_i), if no other process is active, otherwise, it is set to the finish time of the last job of which priority is higher than τ_i's. The outermost cycle uses the middle cycle to determine if the job is schedulable. If not, then it terminates the algorithm returning failure. Otherwise, it tests the next job, until the loading effects are dissipated, i.e., until a job does not load the following one. If such tests end without any job's deadline is violated, then it returns success.

The middle cycle starts by determining the duration of T^{L}. It adds this amount to the c' until it becomes equal to $\tau_i.c$, but not greater since in this case the job would finish, hence $c' \leftarrow \min\{\tau_i.c, c'+r\}$. It updates the current time accordingly and checks for deadline violations. Then it uses the innermost cycle to determine the duration of the next T^{HE}, which it skips.

The innermost cycle simply repeats itself for as long as there are higher priority jobs in the execution queue, in order to determine the finish time of the current T^{HE}.

Due to its multiple cycles, Algorithm 1 has an high computational complexity, not being suitable for online use. However, despite the dynamic nature of the algorithm, it can be executed offline, since all the information necessary for the scheduling will be already available. Such offline scheduling produces a scheduling table that can be used online to choose the jobs that get executed and it does not have an high online computational requirement.

An example of the execution of Algorithm 1, in which the queue is EDF sorted, is provided in Figure 2. Algorithm 1 is used to test for schedulability the introduction of τ_4. In $t = t_1, \tau_4$ is released, thus Algorithm 1 is executed. At this point τ_5 is in execution, but since τ_5 has a priority lower than that of τ_4, the algorithm marks the period between t_1 and t_2 as belonging to τ_4's T^{L} and sets $c' \leftarrow t_2 - t_1$. It runs the

Algorithm 1 Inserting an optional job into the execution queue

$i \leftarrow$ index of job to be inserted
$T^{HE} \leftarrow$ set of jobs with higher priority than τ_i
$T^{L} \leftarrow$ set of jobs with equal or lower priority than τ_i
$T \leftarrow$ set of all jobs inside the execution queue (jobs are sorted by inverse priority)
τ_j %%j^{th} job in T
$\tau_j.\{f, d, r, c\}$ %%j^{th} job's finish, deadline, release and WCET times
$m \leftarrow SYS_{TIME}$ %%current time
repeat
  $m \leftarrow \max\{\tau_i.r, \tau_{i-1}.f\}$ %%initializes current time
  $c' \leftarrow 0$ %%amount of CPU time hitherto used by the current job
  repeat
    $r \leftarrow \min\{\tau_j.r | (\tau_j.f > m \wedge j \leq i)\} - m$ %%execution time of tasks in T_L
    $c' \leftarrow \min\{\tau_i.c, c'+r\}$ %%add amount of time that the job can execute before being preempted
    $c' \leftarrow \min\{\tau_i.c, c'+r\}$ %%add amount of time that the job can execute before being preempted
    $m \leftarrow \min\{\tau_i.c - c', r\} + m$ %%update current time
    if $m > \tau_i.d$ **then**
      return FAIL %% deadline violated! Return fail
    end if
    if $c' == \tau_i.c$ **then**
      $\tau_i.f \leftarrow m$ %%the job's finish time is equal to the current time
    else
      $T_1 \leftarrow \{\forall_k \tau_k : (\tau_k.r == m) \wedge (\tau_k.d < \tau_i.d)\}$ %%set of all jobs that are released at this instant and belong to T^{HE}
      repeat
        $c\_ \leftarrow \sum T_1.c$ %%total amount of time required by jobs in T^{HE}
        $T_2 \leftarrow \{\forall_k : m < \tau_k.r < m + c\_ \wedge \tau_k.d < \tau_i.d\}$ %%set of previously released jobs that belong to T^{HE}
        $m \leftarrow m + c\_$ %%update current time
        if $m > \tau_i.d$ **then**
          return FAIL %%if deadline is past, then return fail
        end if
      until $T_2 == \{\}$ %%until the queue only have lower priority jobs
    end if
  until $c' == \tau_i.c$ %%until the completion of the job
  $i \leftarrow i + 1$ %%test next job
until $\tau_{i-1}.f \leq \tau_i.r$ %%the last job finishes before the start of the current
return SUCCESS

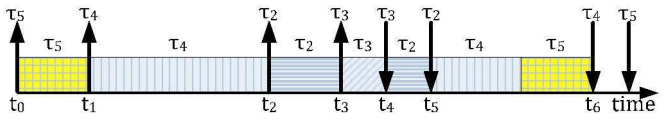

Fig. 2: Example of execution a test of a single job as described by Algorithm 1.

inner most cycle, which determine that the interval t_2 to t_5 is in T^{HE}. Similarly, the interval from t_5 to t_6 is a T^L, but the execution of τ_4 ends before t_6 because it reaches its WCET, a fact that is accounted for by the line $c' \leftarrow \min\{\tau_i.c, c' + r\}$ in which, in this particular case $r \leftarrow t_6 - t_5$.

t_5 also needs to be checked for schedulability, since it has a priority lower than that τ_4. In this case, c' is initialized with $t_1 - t_0$, which is the amount of CPU time that t_5 had already used. Moreover, the current time is set to the instant in which t_4 ended. By virtue of being the only job in execution, t_5 is the highest priority job, executes until successful completion at t_6.

B. Precedence Rules

In the last sections it was shown that 1) frames can be changed dynamically without causing a dynamic failure and 2) an algorithm to insert an optional job into the execution queue was presented. However, if all tasks that can independently enter the execution queue without causing a deadline miss do so simultaneously, it might happen that the new execution queue/hiperframe becomes not schedulable. Hence, in order to guarantee that mandatory jobs will not fail their deadlines, it is not sufficient to test for each job at a time, but it is necessary to test for all optional jobs trying to be executed.

At this point a number of new concepts must be introduced to help understand when a deadline might be failed. First, as mentioned above, all jobs that get executed are put in a single execution queue. This is a significant departure from approaches that use two queues, one for mandatory jobs and another for optional jobs, with jobs in the second queue being executed if and only if the first queue is empty. The reason for a single queue is that 1) it completely excludes the possibility of an optional task starting its execution but missing its deadline and 2) it will never happen that an optional task could be scheduled if it were executed before a mandatory task but it is not schedulable if executed afterwards, thus decreasing the number of executed optional tasks.

The EDF scheduling policy is used due to its high utilization and relatively low computational demand, when compared to other dynamic priorities protocols. Furthermore, EDF can be considered a static priority assignment policy at the job level, meaning that any two jobs in an hiperframe, with phase relations that allow them to compete for resources, maintain their priority relationship. Thus, EDF is a natural choice for dynamic scheduling that involves changes to the (hiper)frame. Other aspect that favors EDF is its associated large body of knowledge.

Hence, Algorithm 1 is EDF based. Nevertheless, as stated before, there may be more than one job that performs this action and concludes that it is eligible to enter the queue. To solve this problem, a precedence rule is introduced and only the highest precedence jobs are inserted in the execution queue. Optional jobs try to enter in the execution queue by an order defined by the precedence rules. Lower precedence jobs will find the queue already full. Good candidates for precedence mechanisms are those that choose optional jobs that increase the overall schedulability, or simply that increase the number of future optional jobs that will be executed. However, to the best of the author's knowledge, there is no computationally light algorithm that allows to perform such type of ordering in a reasonable amount of time. Hence, it was adopted a simple ordering mechanism that consists in serving the optional jobs with a priority that is directly proportional to the time of the last won arbitration.

Note that the precedence rules are internal to the scheduler. For example, it would be tempting to choose, in a control setting with a draw concerning optional jobs, the job associated with a task that had the highest value of some monotonically increasing function of the error. However, 1) by definition these draws are rare and 2) as seen from the discussion above, the scheduler has to choose the loser of the previous draw in order to maintain schedulability. It is in this sense that these draws are internal to the scheduler. The only draws that do not have these characteristic are the first to happen. However, also by definition, these are even more scarce, too scarce to have a significant impact.

C. Shifting Mandatory Jobs

As presented in Section IV-A, frame changes were bound to happen only in optional jobs that actually got executed. However, there are task sets in which:

- the task set is schedulable,
- it is impossible to stop jobs from all tasks from being placed as mandatory and released simultaneously,
- whenever jobs from all tasks are released simultaneously at least one deadline is missed.

Under such conditions, there is no possible shift of the optional jobs that could possibly save the system from missing a deadline. A possible solution is to do with mandatory jobs what is also done with the optional ones, namely:

- whenever mandatory jobs are released, check if all mandatory jobs can be executed,
- if not, sort them according to a predefined set of parameters and the top jobs are executed as mandatory, the rest act as optional jobs,
- if a mandatory job is not executed, the leftmost bits of its (m, k)–firm frame are rotated.

The second point above should be done with a certain amount of caution since, unlike in the optional case, if done repeatedly will lead to a dynamic failure. To overcome this problem, an initial schedulability analysis should be performed. In that way it would be guaranteed that all such (m, k)–firm frame transformations would lead to systems

that are still (m,k)–firm schedulable. Nonetheless, an online approach, which still requires an initial schedulability analysis is provided, based on the following theorem:

Theorem 4. *Consider a set of schedulable* (m,k)–firm *tasks. The non-execution of a mandatory job of one of those tasks does not cause a violation of the* (m,k)–firm *guarantees of that task if during its previous* $k-1$ *job releases the task had at least* m *executions and its rightmost bit is kept fixed, while the remaining bits are rotated one bit to the left.*

Proof: As in the case of optional jobs, execution-wise, a mandatory job that did not get executed is akin to an optional job that also did not get executed. Therefore, after changing the last bit from $0 \rightarrow 1$, and shifting the leftmost $k-1$ bits, the job execution will appear as presented in Figure 3. The sequences before the transition will be scheduled to the (m,k)–firm schedulability assumption, the sequence of k consecutive bits starting at b_0 has m execution due to the assumption that its previous $k-1$ job release (from b_0 to b_{n-1}) had at least m executions. ■

It is noteworthy that the main condition for the previous theorem is met if and only if the task had executed an optional job in its recent past, i.e., within the last $k-1$ jobs, otherwise the number of jobs activations within the last $k-1$ jobs would be $m-1$. Another noteworthy aspect of the last theorem is the fact that it applies only to one task. The case of a group of tasks will be developed in the following sections.

As in the optional job case, precedences can/must also be defined, which allow the scheduler to choose among the mandatory jobs that can be turned into optional ones, i.e. the jobs *left out* in case of temporary overloads. The use of a precedence rule proved itself always successful in tests that were performed by the authors. Some of these tests are presented in Section V. In fact, this is a result that the author has not been able to prove: the need for shifting a mandatory job only occur in the first jobs of a task, either at the beginning of the executions or when a new task is introduced in the task set.

The last three paragraphs suggest an alternative view of the scheduler presented herein, namely

1 initialize the execution set with k bits in which m bits are set,
2 if the non-execution of a job would jeopardize the (m,k)–firm guarantees (among the last $k-1$ jobs only $m-1$ jobs were executed), set it as mandatory. Set it as optional otherwise,

$$\overbrace{[b_{n-1}\, b_{n-2}\, b_{n-3} \cdots b_2\, b_1 \underbrace{\overbrace{b_0}^{rep\rightarrow}}_{b_0=1} b_0\, b_{n-2}\, b_{n-3} \cdots b_1\, b_{n-1}\, b_0]}^{\mathrm{m+1\,executions}}$$

Fig. 3: State of (m,k)–firm frames during the non execution of a mandatory job

3 optional jobs compete for the execution in unused *spaces*,
4 if a draw occur, when the precedence rule is applied, the task that won the last arbitration loses the current one,
5 if a draw remains, choose jobs for execution randomly.

D. Hiperframe Initialization

Dynamic (m,k)–firm scheduling has a number of peculiarities that are nonexisting in static (m,k)–firm scheduling. For instance, in the static case the hiperframe is known *a priori*, i.e., before the activation of the first job. A naïve approach would be to initialize the dynamic scheduler with a statical hiperframe that was known to have reasonable scheduling characteristics. But the knowledge of such hiperframe implies some sort of static scheduling, which defeats the purpose of using a dynamic scheduler.

In spite of this aspect, the dynamic approach is still of use, because if the task set is schedulable then the dynamical approach converges to an hiperframe that is schedulable regardless of the initial hiperframe. However, it is possible that there is more than one possible schedule that verifies the (m,k)–firm conditions, some of them differing only in their phases. Moreover, different initial hiperframes may converge to different final hiperframes.

Regarding the initialization *per se*, as stated before, it is not known *a priori* any schedulable hiperframe. More precisely, for operations like the postponement of a mandatory job, as previously discussed, it is vital to already know the (m,k)–firm frame of the job in question. Therefore, measures are necessary to ensure that such initial lack of knowledge will not turn into a dynamic failure in an otherwise (m,k)–firm schedulable set.

One way of achieving such goal is to assume that all previous jobs, of tasks that have not yet activated any jobs, were successfully executed. This ensures that at the beginning of the hiperframe all possible transformations are allowed. However, from an execution point-of-view, being able to execute such jobs even while having to change the initial hiperframe is akin to start from a correct hiperframe and execute the same jobs. Hence, by assuming that all jobs before the beginning of the hiperframe were executed the hiperframe will automatically act as or converge to a *correct* hiperframe.

The strategy presented in the last paragraph is, in a sense, optimal but not unique. For example, the phases of the hiperframe can be used to develop several variations of the presented strategy.

V. Examples of Dynamic Schedules

This section presents a few examples of the execution of the presented scheduler. These examples show some emerging properties of the dynamic scheduler. (m,k)–firm tasks will be described as (C,T,m,k) were the first two variables designate the *worst case execution time* and the *period*, respectively.

The first example is the task set $T_1 = \{(5,6,1,3),(4,5,1,2)\}$. Its schedule is represented in Figures 4 and 5. A static (m,k)–firm scheduler would have problems scheduling this simple task set since, regardless of

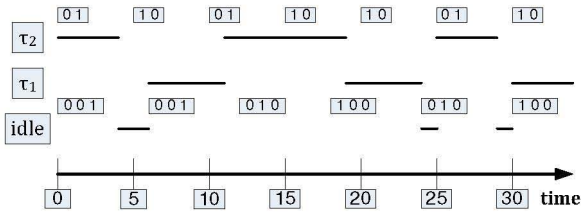

Fig. 4: First half of a possible schedule of T_1 generated by the dynamic scheduler proposed in this document

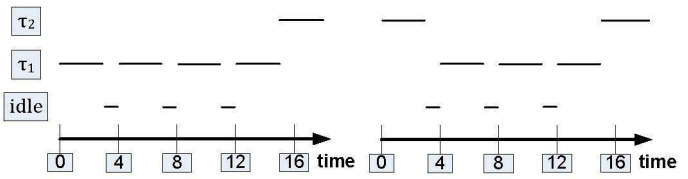

Fig. 6: Schedule of task set of T_2 generated by GDPA — left starts with job from τ_1, right starts with job from τ_2

the initial phases (both of the tasks and of their hiperframes), there would be an instant in which both tasks would generate mandatory jobs. Once more, the initial hiperframes are unknown, hence it is assumed that all previous activations were executed. Let both jobs start as mandatory at instant $t = 0$ and let the job from τ_2 be chosen for execution. It will end its execution at instant $t = 4$. At $t = 5$ the second job from τ_2 is activated. Since it is optional and its execution would jeopardize the execution of the second job of τ_1, which is mandatory, it is not executed. Hence, this optional job does not get executed even tough the execution queue is currently empty. At $t = 6$ the second job of τ_1 is activated and executed. At $t = 10$, the third job of τ_2 is activated as mandatory and starts its execution at $t = 11$ after the completion of the job from τ_1. At $t = 12$, an optional job of τ_1 is released, which is also not executed because it would not meet its deadline due to interference of the ongoing job. At $t = 15$ another optional job from τ_2 is activated and executed. At $t = 18$, τ_1 activates an optional job that gets executed because the next job from τ_2 (activated at $t = 20$) is also optional and only one of them can be activated. Using the rules defined above, there is a draw and τ_2 executed last. Continuing, the next job activation (τ_1) occurs at $t = 24$, which is an optional job that cannot be executed because at $t = 25$ a mandatory job (τ_2) will be activated. The later job ends its execution at $t = 29$ and at $t = 30$ the whole process repeats itself. Note that the scheduling generated for τ_1 a (m,k)–firm frame $[0\,1\,0\,1\,0]$ and for τ_2 $[1\,0\,1\,1\,0\,1]$ or simply $[1\,0\,1]$ (internal period). This schedule is represented at figures 4 and 5. Note also that if in the moment at which the two optional jobs were competing for execution ($t = 18$) τ_1 were chosen for execution then the (m,k)–firm frames would be τ_1 $[0\,1\,0\,0\,1]$ and τ_2 $[1\,0\,1\,1\,1\,0]$. There is also another possibility that emerges by starting the scheduling by choosing the job from the first task as the first job to be executed, which would lead to: τ_1 $[1\,0\,0\,1\,0]$ and τ_2 $[0\,1\,1\,1\,0\,1]$ or τ_1 $[1\,0\,1\,0\,0]$ and τ_2 $[0\,1\,1\,0\,1\,1]$ (or $[0\,1\,1]$), according to a particular choice of optional job. Once again, the second hiperframe is comprised of rotated (m,k)–firm frames of the first hiperframe. In particular the first set can be generated by starting the sequence shown in this example at $t = 6$.

The following example shows that the proposed scheduler is able to schedule task sets that GDPA is not. Consider the task set $T_2 = \{(3,4,1,2),(3,5,1,2)\}$. On this task set, GDPA, regardless of the choice of the first task to be executed would cause τ_2 to have a dynamic failure since τ_1 would, on several occasions, terminate the execution of one of its jobs, wait one time unit and then restart, because jobs from τ_2 would not be in execution, leaving the execution queue empty. On the other hand, several jobs from τ_2 are activated while a job from the first task is in execution. Thus, the execution queue would be full. Figure 6 presents all possible schedules that the GDPA can generate on this task set.

The same task set can be scheduled using the presented scheduler and such schedule is presented in Figure 7. For instance, in $t = 4$ when the second job of τ_1 is activated, it does not enter in the ready queue due to a blocking from the mandatory job from τ_2 (activated at $t = 5$). The rest of the schedule is similar to previous schedule. More specifically, it converges to a suitable hiperframe at $t = 8$, with $\tau_1 : [1\,0\,1\,0\,1]$ and $\tau_2 : [0\,1\,0\,1]$ or $\tau_2 : [0\,1]$ due to the internal period. Once again, starting with a job from τ_2 would only cause a phase difference in the hiperframe.

VI. Comparison with Previous Solutions

The presented scheduler improves upon previous schedulers because it does not require a (k,k)–firm reservation (see Section II). It is an improvement even over the work in [5], which searched over all possible static frames of each task with the corresponding values of m and k, i.e., for each

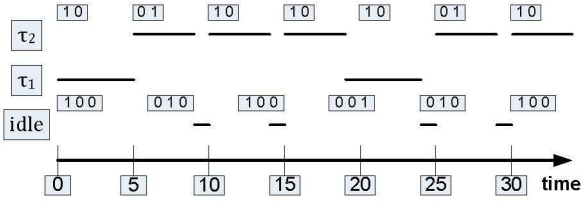

Fig. 5: Second half of a possible schedule of T_1 generated by the dynamic scheduler proposed in this document

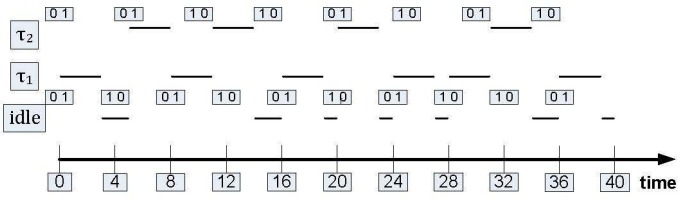

Fig. 7: A possible schedule of T_2 generated by the dynamic scheduler proposed in this document

$(\mathrm{m_i}, \mathrm{k_i})$–firm task it was searched for all possible k_i bits long frames that contained m_i ones. This implies that the presented scheduler outperforms any static scheduler of this family. The following theorem provides a formal proof of this fact.

Theorem 5. *The set of* (m, k)–firm *task sets that can be scheduled by classical static schedulers in which* m *and* k *of the* (m, k)–firm *frames are equal to the respective values of the* (m, k)–firm *task, is a proper subset of the set of* (m, k)–firm *task sets that can be scheduled by the dynamic scheduler presented herein.*

Proof: The theorem is true if all (m, k)–firm task sets that can be scheduled by a static scheduler can also be scheduled by the novel scheduler, i.e., the *subset* part, and there is at least one task set that is schedulable by the proposed scheduler but is not schedulable by any static scheduler, implying *proper subset* part. The *subset* part 1) if there is a static (m, k)–firm schedule that meets all (m, k)–firm guarantees using (m, k)–firm frames with values of m and k equal to the respective values of the (m, k)–firm, then if the dynamic scheduler is initialized with the aforementioned schedule (hiperframe), in the worst case scenario, i.e., no optional job is executed, the dynamic scheduler will send exactly the same sequence of jobs into the execution queue. And 2) as discussed above, regardless of the initial schedule (hiperframe), if the task set is schedulable, then the hiperframe converges into a schedulable (fixed) hiperframe, possibly the static schedule in question. From what follows that if a classical static scheduler can meet all guarantees, so can the dynamic scheduler. For proving the *proper* part it suffices to present an example of a task set that is schedulable by the dynamic scheduler but is not schedulable by any classical static scheduler. To this end consider a task set with two tasks, for which the classical utilization, i.e., $U = \sum_i \frac{C_i}{T_i} > 1$, with periods that are not multiples of one another and with implicit deadlines. Consider also, that their (m, k)–firm properties are chosen in such a that it would allow the dynamic scheduler to meet its deadlines, for instance $m_i = 1$ and $k_i = 2\left\lceil \frac{T_j}{T_i} \right\rceil$, where j represents the other task. Due to the fact that the periods are not multiples of each other, when scheduled by a stationary scheduler, there will be an instant in which both tasks are activated as mandatory. At such instant at least one of the tasks will fail its deadline since the combined utilization of both tasks is greater than one. Therefore no static scheduler can guarantee that the (m, k)–firm constraints will be met. However, this task set is schedulable by the dynamic scheduler by construction. ■

An important remark is that classical static (m, k)–firm schedulers can only schedule task sets that are EDF schedulable (see discussion at the beginning of this section) and all EDF schedulable task sets are schedulable by the dynamical algorithm proposed here, since the execution queue is itself EDF scheduled. Furthermore, besides the point raised by the last theorem about the set of schedulable task sets, even among the task sets for which there is a classical static schedule, the proposed scheduler achieves utilizations that are never inferior to those achieved by classical statical schedulers.

An important difference, which is also an improvement, between the presented approach and classical static (m, k)–firm schedulers, is the fact that in the classical approaches the schedulers have two queues, namely: queue 1, devoted to mandatory jobs, and all jobs that are in it get scheduled; queue 2, is used to schedule optional jobs, and these jobs are only executed if queue 1 is empty. Each queue is scheduled according to classic scheduling policies, such as EDF. In the proposed scheduler, it is used only one queue, which in turn may increase the number of optional jobs that are executed. For example, consider Figure 8, in which the mandatory job has a *far* absolute deadline, whereas there is an optional job with a *close* deadline. In the scheme that is used in classical static/dynamic (m, k)–firm schedulers, the mandatory job would be in the highest priority queue, hence it would be executed first. Thus, the optional job would not be executed right away because the mandatory job was already in execution, leading to a deadline miss. However, in the proposed approach, after certifying itself that the optional job would not cause a deadline miss, the scheduler would execute the optional job, as shown in Figure 8, thus, meeting all deadlines.

But there are also a few downturns. The first is related to the high computational complexity of the proposed scheduler. A related negative aspect, which is common to all dynamic schedulers, deals with the loss of predictability under overload conditions. The loss of predictability occurs in the sense that in some classical (m, k)–firm schedulers, whenever there is a violation of an (m, k)–firm guarantee, it is possible to predict the order in which jobs will miss their deadlines. For instance, in the scheduler of [2], the tasks with the longest periods suffer the violations first, since on such scheduler assumes synchronous task release and that jobs in the execution queue are served in a Rate Monotonic manner. Furthermore, when a job misses a mandatory job it also violates its (m, k)–firm guarantee. But, for dynamic (m, k)–firm schedulers, it is not possible to make similar predictions.

The proposed scheduler improves upon the work of DBP and its variants because it does not have any of the downturns presented in Section II. Examples of such downturns include a disregard for the deadlines of the tasks, the fact that the scheduler tends to schedule tasks with a short distance to failure close to each other, a total disregard for the relation

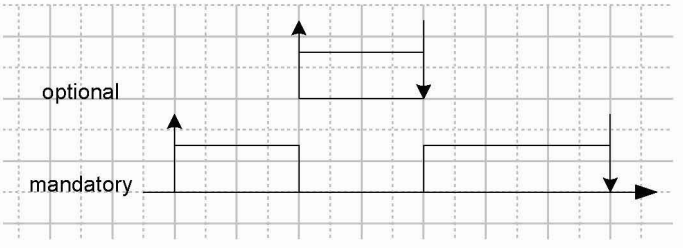

Fig. 8: Example of an advantage of a single queue.

of the periods of the tasks and a disregard for the relations between the time to failure of the various tasks.

A third negative aspect of the use of dynamic (m, k)–firm schedulers is related to the fact that some applications require that, beside the usual (m, k)–firm guarantees, tasks must have a mandatory job in a pre-determined part of the (m, k)–firm frame. For example the JPEG frame transmission, see Section II. Nevertheless, it is possible to make some modifications in order to properly accommodate these requirements, such as splitting the task, giving different values of m and k to each of the subtasks, one or more behaving as a classical task, i.e., executing all of its jobs. Alternatively, it is possible to modify the (m, k)–firm definitions as to have three levels of jobs instead of the current two.

VII. Conclusion

(m, k)–firm schedulers have a great potential to improve the performance of many real-time applications. However, many applications require deterministic guarantees regarding the performance of the schedulers.

Traditionally, dynamic schedulers have been mostly comprised by probabilistic schedulers of the DBP family. A novel deterministic dynamic scheduler is presented in this paper. The proposed dynamic scheduler hinged upon the fact that it is possible to dynamically change an (m, k)–firm frame while maintaining its respective guarantee. In fact, there are task sets in which is possible to provide the respective guarantees only if the respective frame is changed at some point. Therefore, this scheduler is able to schedule (m, k)–firm task sets that are not schedulable by conventional deterministic (m, k)–firm schedulers, as well as to improve the resource utilization of task sets that are schedulable by those conventional (m, k)–firm schedulers.

Notwithstanding the novelty of this new scheduler, some points remain to be optimized. For example, the algorithm for the insertion of a new job on the execution queue, in its current state is computationally intensive, implying in practice that the scheduler must be used offline. Another point that needs to be investigated is the schedulability test.

References

[1] L. Jian and S. Ye-Qiong, "Relaxed (m, k)-firm constraint to improve real-time streams admission rate under non pre-emptive fixed priority scheduling," in *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*, sept. 2006, pp. 1051 –1060.

[2] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 10, no. 6, pp. 549 –559, jun 1999.

[3] G. Bernat and R. Cayssials, "Guaranteed on-line weakly-hard real-time systems," in *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, dec. 2001, pp. 25 – 35.

[4] J. Lin and A. Cheng, "Maximizing guaranteed qos in (m, k)-firm real-time systems," in *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on*, 0-0 2006, pp. 402 –410.

[5] G. Quan and X. Hu, "Enhanced fixed-priority scheduling with (m,k)-firm guarantee," in *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE*, 2000, pp. 79 –88.

[6] G. Koren and D. Shasha, "Skip-over: algorithms and complexity for overloaded systems that allow skips," in *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, 1995, pp. 110–117.

[7] L. Niu and G. Quan, "Energy minimization for real-time systems with (m,k)-guarantee," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 7, pp. 717–729, 2006.

[8] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *Computers, IEEE Transactions on*, vol. 44, no. 12, pp. 1443 –1451, dec 1995.

[9] K.-I. Kim, "A novel scheduling for (m, k)-firm streams in wireless sensor networks," in *Networked Computing and Advanced Information Management (NCM), 2010 Sixth International Conference on*, aug. 2010, pp. 553 –556.

[10] K. H. Kim, J. Kim, and S. J. Hong, "Best-effort scheduling (m, k)-firm real-time tasks based on the (m, k)-firm constraint meeting probability." in *ESA/VLSI'04*, 2004, pp. 240–248.

[11] J.-m. C. Zhi Wang and Y. xian Sun, "An integrated dbp for streams with (m,k)-firm real-time guarantee," *JOURNAL OF ZHEJIANG UNIVERSITY*, vol. 5, pp. 816–826, 2004.

[12] J. Chen, Y. Song, Z. Wang, and Y. Sun, "Equivalent matrix dbp for streams with (m,k)-firm deadline," in *Industrial Electronics, 2004 IEEE International Symposium on*, vol. 1, may 2004, pp. 675 – 680 vol. 1.

[13] L. Lanying and T. Yu, "Analysis and improvement of scheduling algorithms based on (m, k)-firm constraint," in *Computer Science and Computational Technology, 2008. ISCSCT '08. International Symposium on*, vol. 2, dec. 2008, pp. 74 –77.

[14] Y. Kong and H. Cho, "Guaranteed scheduling for (m,k)-firm deadline-constrained real-time tasks on multiprocessors," in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2011 12th International Conference on*, 2011, pp. 18–23.

[15] B. Li and K.-I. Kim, "A novel routing protocol for (m, k)-firm-based real-time streams in wireless sensor networks," in *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, 2012, pp. 1715–1719.

[16] C. Evequoz, "Guaranteeing optional task completions on (m,k)-firm real-time systems," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, 2010, pp. 1772–1779.

[17] H. Cho, Y. Chung, and D. Park, "Guaranteed dynamic priority assignment schemes for real-time tasks with (m, k)-firm deadlines," *ETRI Journal*, vol. 32, pp. 422–429, june 2010.

[18] P. Ramanathan, "Graceful degradation in real-time control applications using (m, k)-firm guarantee," in *Fault-Tolerant Computing, 1997. FTCS-27. Digest of Papers., Twenty-Seventh Annual International Symposium on*, jun 1997, pp. 132 –141.

[19] J. Li, Y. Song, and F. Simonot-Lion, "Providing real-time applications with graceful degradation of qos and fault tolerance according to -firm model," *Industrial Informatics, IEEE Transactions on*, vol. 2, no. 2, pp. 112 – 119, may 2006.

[20] F. Flavia, J. Ning, F. Simonot-Lion, and S. YeQiong, "Optimal on-line (m,k)-firm constraint assignment for real-time control tasks based on plant state information," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, sept. 2008, pp. 908 –915.

[21] T. Wu and S. Jin, "Weakly hard real-time scheduling algorithm for multimedia embedded system on multiprocessor platform," in *Ubi-Media Computing, 2008 First IEEE International Conference on*, 31 2008-aug. 1 2008, pp. 320 –325.